Holland & Knight

Tel (617)523-2700
Fax (617)523-6850

Holland & Knight LLP
10 St. James Avenue
11th Floor
Boston, MA 02116
www.hklaw.com

TO:

NAME	COMPANY/FIRM	FAX NUMBER
Jeffrey Swearingen	USPTO	571-273-3921
CITY/STATE		TELEPHONE NUMBER
		571-272-3921

FROM:

NAME	TELEPHONE
Seth Milman	(617) 305-2136
DATE	TOTAL PAGES (Including Cover Sheet)
12/14/2009 10:15:50 AM	17

If you did not receive all of the pages or find that they are illegible, please call **(617) 305-2136**

MESSAGE: Dear Examiner Swearingen:

Here are **draft claims from the client in application no. 10/069,670. The amendments reflect changed proposed on Dec. 10, 2009, namely, including the last element of claim 4 in claims 22, 25, and 26. The claims also include amendments faxed to your attention on Dec. 3, 2009.**

Respectfully submitted,

Seth Milman

Reg. no. 64,573

Draft claims
Application No. 10/069,670
H&K Docket No. 102114.00034

Proposed Listing of Claims:

1-3. (Canceled)

4. (Currently Amended) A method of verifying a program fragment downloaded onto a reprogrammable embedded system, equipped with a rewritable memory, a microprocessor and a virtual machine equipped with an execution stack and with operand registers, said program fragment consisting of an object code and including at least one subprogram consisting of a series of instructions manipulating said operand registers, said microprocessor and virtual machine configured to interpret said object code, said embedded system being interconnected to a reader, wherein subsequent to a detection of a downloading command and a storage of said object code in said rewritable memory, said method, for each subprogram, comprises:

initializing a type stack and a table of register types through data representing a state of said virtual machine on initialization of an execution of said temporarily stored object code;

carrying out a verification process of said temporarily stored object code instruction by instruction, by discerning an existence, for each current instruction, of a target, a

branching-instruction target, a target of an exception-handler call or a target of a subroutine call, and, said current instruction being a target of a branching instruction, said verification process including verifying that said type stack is empty and rejecting said program fragment otherwise;

verifying and updating an effect of said current instruction on the data types of said type stack and of said table of register types including:

verifying that said type stack includes at least as many entries as said current instruction includes operands;

unstacking and verifying that types of entries at the top of said stack are subtypes of the types of said operands of said current instruction; and

stacking data types which are assigned to said results on said stack;

said verification process being successful when said table of register types is not modified in the course of a verification of all said instructions, and said verification process being carried out instruction by instruction until said table of register types is stable, with no modification having taken place, said verification process being interrupted and said program fragment being rejected, otherwise.

5. (Previously Presented) The method of claim 4, wherein the types which are manipulated during said verification process include at least:

class identifiers corresponding to object classes which are defined in said program fragment;

numeric variable types including at least a type for an integer coded on a given number of bits, designated as short type, and a type for the return address of a jump instruction, designated as a return address type;

references of null objects designated as null type;

object type relating to objects designated as object type;

a first specific type representing an intersection of all said types and corresponding to a zero value, designated as an intersection type;

a second specific type representing a union of all said types and corresponding to any type of value, designated as a union type.

6. (Previously Presented) The method of claim 5, wherein all said types obey a subtyping relation:

said object type is a subtype of said union type;

said short type and said return address type are subtypes of said union type; and

said intersection type is a subtype of the null type, the short type and the return address type.

7. (Canceled)

8. (Previously Presented) The method of claim 4, wherein said current instruction being a target of a subroutine call, said verification process comprising:

verifying that a previous instruction to said current instruction is an unconditional branching, a subroutine return or a withdrawal of an exception; and reupdating said type stack by an entity of the return address type, formed by the return address of said subroutine, in case of a positive verification process; and,

rejecting said program fragment in case said verification process is failing, otherwise.

9. (Previously Presented) The method of claim 4, wherein said current instruction being a target of an exception handler, said verification process comprising:

verifying that a previous instruction to said current instruction is an unconditional branching, a subroutine return or a withdrawal of an exception; and

reupdating said type stack, by entering an exception type, in case of a positive verification process; and

rejecting said program fragment in case of said verification process is failing.

10. (Previously Presented) The method of claim 4, wherein said verification process fails and said program fragment is rejected if said current instruction is a target of multiple incompatible branchings.

11. (Previously Presented) The method of claim 4, wherein said verification process comprises continuing by passing to an update of said type stack if said current instruction is not a target of any branching.

12. (Currently Amended) The method of claim 4, wherein said verifying and updating includes[[,]] ~~at least~~:

~~verifying that said type stack includes at least as many entries as said current instruction includes operands;~~

~~unstacking and verifying that types of entries at the top of said stack are subtypes of the types of said operands of said current instruction;~~

verifying an existence of a sufficient memory space on said type stack to proceed to stack the results of said current instruction[[;]],

~~stacking data types which are assigned to said results on said stack.~~

13. (Previously Presented) The method of claim 12, wherein said current instruction being an instruction to read a register of a given address, said verification process comprises:

verifying the data type of the result of a corresponding reading, by reading an entry at said given address in said table of register types;

determining said effect of said current instruction on said type stack by unstacking the entries of said type stack corresponding to the operands of said current instruction and by stacking said data type of said result.

14. (Previously Presented) The method of claim 12, wherein said current instruction being an instruction to write to a register of a given address, said verification process comprises:

determining an effect of said current instruction on said type stack and the given type of the operand which is written in this register at said given address;

replacing the type entry of said table of register types at said given address by the type immediately above said previously stored type and above said given type of said operand which is written in said register at said given address.

15-19. (Canceled)

20. (Currently Amended) An embedded system which can be reprogrammed by downloading program fragments, said embedded system including a least one microprocessor, one

random-access memory, one input/output module, one electrically reprogrammable nonvolatile memory and one permanent memory, an installed main program and a virtual machine adapted to execute said installed main program and at least one program fragment using said microprocessor, wherein said embedded system includes at least one verification program module to verify a downloaded program fragment in accordance with a process comprising:

initializing a type stack and a table of register types through data representing a state of said virtual machine at a starting of an execution of said temporarily stored object code;

carrying out a verification process of said temporarily stored object code instruction by instruction, by discerning an existence, for each current instruction, of a target, a branching-instruction target, a target of an exception-handler call or a target of a subroutine call, and, said current instruction being a target of a branching instruction, said verification process consisting of verifying that said type stack is empty and rejecting said program fragment otherwise;

carrying out a verification process and updating of an effect of said current instruction on the data types of said type stack and of said table of register types;

verifying and updating an effect of said current instruction on the data types of said type stack and of said table of register types including:

verifying that said type stack includes at least as many entries as said current instruction includes operands;

unstacking and verifying that types of entries at the top of said stack are subtypes of the types of said operands of said current instruction; and

stacking data types which are assigned to said results on said stack;

said verification process being successful when said table of register types is not modified in the course of a verification of all of said instructions, and said verification process being carried out instruction by instruction until said table of register types is stable, with no modification having taken place, said verification process being interrupted and said program fragment being rejected, otherwise;

said verification program module being installed in said permanent memory.

21. (Canceled)

22. (Currently Amended) A system for transforming an object code of a program fragment including a series of instructions, in which operands of each instruction belong to data types manipulated by said instruction, an execution stack does not exhibit any overflow phenomenon and for each branching instruction, the types of stack variables at said branching are identical to the types of stack variables at targets of this branching, and an operand, of given type, written to a register by an instruction of said object code is reread from the same register by another instruction of said object code with the same given data type, into a standardized object code for said program fragment, wherein said

transforming system includes, at least, installed in a memory of a development computer or workstation, a program module for transforming said object code into a standardized object code in accordance with a process of transforming including for all instructions of said object code comprising:

annotating each current instruction with the data type of said type stack before and after execution of said current instruction, with an annotation data being calculated by means of analysis of the data stream relating to said current instruction;

detecting, within said instructions and within each current instruction, an existence of branchings, or respectively of branching-targets, for which said execution stack is not empty, said detecting operation being carried out on the basis of said annotation data of said type of stack variables allocated to each current instruction; and, in case of detection of a non-empty execution stack,

inserting instructions to transfer stack variables on either side of each said branching or branching target respectively, in order to empty contents of said execution stack into temporary registers before said branching and to reestablish said execution stack from said temporary registers after said branching; ~~and~~

not inserting any transfer instruction otherwise, this method allowing thus to obtain said standardized object code for said program fragment, in which said operands of each instruction belong to the data types manipulated by said instruction, said execution stack does not exhibit any overflow phenomenon, said execution stack is empty at each branching instruction and at each branching-target instruction, in the absence of any modification to the execution of said program fragment; and

verifying and updating an effect of said current instruction on the data types of said type stack including:

verifying that said type stack includes at least as many entries as said current instruction includes operands;

unstacking and verifying that types of entries at the top of said stack are subtypes of the types of said operands of said current instruction; and

stacking data types which are assigned to said results on said stack; and

carrying out a verification process, said verification process being successful when a table of register types is not modified in the course of a verification of all said instructions, and said verification process being carried out instruction by instruction until said table of register types is stable, with no modification having taken place, said verification process being interrupted and said program fragment being rejected, otherwise.

23. (Canceled)

24. (Currently Amended) A computer program product which is recorded on a computer readable storage medium and can be loaded directly from a terminal into an internal memory of a reprogrammable embedded system equipped with a microprocessor and a rewritable memory, said embedded system making it possible to download and temporarily store a program fragment consisting of an object code including a series of instructions, executable by said microprocessor by way of a virtual machine equipped

with an execution stack and with operand registers manipulated via said instructions and making it possible to interpret said object code, said computer program product including portions of object code to execute at least one of steps of verifying a program fragment downloaded onto said embedded system according to a verifying process, said verifying process comprising:

initializing a type stack and a table of register types through data representing a state of said virtual machine at initialization of execution of said temporarily stored object code;

carrying out a verification process of said temporarily stored object code instruction by instruction, by discerning an existence, for each current instruction, of a target, a branching-instruction target, a target of an exception-handler call or a target of a subroutine call, and, said current instruction being a target of a branching instruction, said verification process consisting in verifying that said execution stack is empty and rejecting said program fragment otherwise;

carrying out a verification process and an updating of an effect of said current instruction on the data types of said type stack and of said table of register types; and

verifying and updating an effect of said current instruction on the data types of said type stack and of said table of register types including:

verifying that said type stack includes at least as many entries as said current instruction includes operands;

unstacking and verifying that types of entries at the top of said stack are subtypes of the types of said operands of said current instruction; and

stacking data types which are assigned to said results on said stack;

said verification process being successful when said table of register types is not modified in the course of a verification of all said instructions, and said verification process being carried out instruction by instruction until said table of register types is stable, with no modification having taken place, said verification process being interrupted and said program fragment being rejected, otherwise.

25. (Currently Amended) A computer program product which is recorded on a computer readable storage medium including portions of object code to execute steps of a process of transforming an object code of a downloaded program fragment into a standardized object code for said program fragment, said process of transforming comprising:

annotating each instruction with a data type of a stack before and after execution of said current instruction, with said annotation data being calculated by means of an analysis of a data stream relating to said current instruction;

detecting, within said instructions and within each current instruction, an existence of branchings, or respectively of branching [[-]] targets, for which an execution stack is not empty, said detecting operation being carried out on based on said annotation data of a type of stack variables allocated to each current instruction, and, in case of detection of a non-empty execution stack;

inserting instructions to transfer stack variables on either side of each said branching or branching target respectively, in order to empty contents of said execution stack into temporary registers before said branching and to reestablish the execution stack from said temporary registers after said branching; ~~and~~

not inserting any transfer instruction otherwise, this method allowing thus:

to obtain said standardized object code for said program fragment, in which the operands of each instruction belong to the data types manipulated by said instruction, said execution stack does not exhibit any overflow phenomenon, said execution stack is empty at each branching instruction and at each branching—target instruction, in absence of any modification to an execution of said program fragment; and

to verify any update an effect of said current instruction on a data type of a type stack including:

verifying that said type stack includes at least as many entries as said current instruction includes operands;

unstacking and verifying that types of entries at the top of said stack are subtypes of the types of said operands of said current instruction;

stacking data types which are assigned to said results on said stack; and

to carry out a verification process, said verification process being successful when a table of register types is not modified in the course of a verification of all said instructions, and said verification process being carried out instruction by instruction until said table of register types is stable, with no

modification having taken place, said verification process being interrupted and said program fragment being rejected, otherwise.

26. (Currently Amended) A computer program product which is recorded on a computer readable storage medium and can be used in a reprogrammable embedded system, equipped with a microprocessor and a rewritable memory, said reprogrammable embedded system allowing to download a program fragment consisting of an object code, a series of instructions, executable by said microprocessor of said reprogrammable embedded system by means of a virtual machine equipped with an execution stack and with local variables or registers manipulated via instructions and making it possible to interpret said object code, said computer program product comprising:

program resources which can be read by said microprocessor of said embedded system via said virtual machine, to command execution of a procedure for managing a downloading of a downloaded program fragment;

program resources which can be read by said microprocessor of said embedded system via said virtual machine, to command execution of a procedure for verifying, by instruction, said object code which makes up said program fragment including:

program resources for verifying and updating an effect of said instruction on a data type of a type stack and of a table of register types including:

verifying that said type stack includes at least as many entries as said current instruction includes operands;

unstacking and verifying that types of entries at the top of said type stack are subtypes of the types of said operands of said current instruction; and

stacking data types which are assigned to said results on said stack; and

program resources for carrying out a verification process, said verification process being successful when said table of register types is not modified in the course of a verification of all said instructions, and said verification process being carried out instruction by instruction until said table of register types is stable, with no modification having taken place, said verification process being interrupted and said program fragment being rejected, otherwise;

program resources which can be read by said microprocessor of said embedded system via said virtual machine, to command execution of a downloaded program fragment subsequent to or in the absence of a conversion of said object code of said program fragment into a standardized object code for this same program fragment.

27. (Previously Presented) The computer program product as claimed in claim 26, additionally including at least one program resource which can be read by said microprocessor of said embedded system via said virtual machine, to command inhibition of execution, by said embedded system, of said program fragment in the case of an unsuccessful verification procedure of said program fragment.

28. (Previously Presented) The method of claim 4 wherein, when said current instruction is the target of a branching instruction, said verification method includes:

verifying that said type stack is empty;

continuing executing of the following instruction if said type stack is empty; and

rejecting the program segment if said type stack is not empty.

9009319_v2